Занятие 8

29.03.2021

**Задание 1**

Бинарные деревья размещаются в обычных массивах: первый элемент - корень, 2\*i+1 это левое поддерево, 2\*i+2 - правое.

*Двоичное дерево поиска* с поддержкой операций:

* *Добавление;*
* *Обход;*
* *Поиск.*

Требование к реализации:

*На базе массива.*

*На базе двухсвязного списка. (22.03.2021)*

**Задание 2**

Два типа данных *Двоичное дерево поиска.* Организуйте код так, чтобы клиент не зависел от реализации.

*Tree.h*

template<class T>  
class Tree {  
 public:  
 virtual ~Tree();  
 virtual void Insert(T key);  
 virtual bool Search(T key);  
 virtual void Print();

};

*TreeDlist.h*

#include <iostream>  
#include "Tree.h"  
template<class T>  
class TreeDlist: public Tree<T> {  
 public:  
 TreeDlist():root\_(0){};  
 ~TreeDlist()override{  
 Node::DestroyNode(root\_);  
 }  
  
 void Insert(T key) override{  
 Insert(root\_,key);  
 };  
 bool Search(T key)override{  
 if(Search(root\_,key)){  
 return true;  
 }  
 return false;  
 };  
 void Print()override{  
 Print(root\_);  
 };  
  
  
 private:  
 struct Node {  
 public:  
 Node(int key):key\_(key){};  
  
 static void DestroyNode(Node\*& node) {  
 if (node) {  
 DestroyNode(node->left\_);  
 DestroyNode(node->right\_);  
 delete node;  
 }  
 }  
  
 T key\_;  
 Node\* left\_= nullptr;  
 Node\* right\_= nullptr;  
 };  
  
  
 void Insert(Node\*& node,T key) {  
 if (!node) {  
 node = new Node(key);  
 } else if (key < node->key\_) {  
 Insert(node->left\_, key);  
 } else if (key > node->key\_) {  
 Insert(node->right\_, key);  
 } else *//key==root->key\_)* return;  
 }  
  
 void Print(Node\* node) {  
 if (node == nullptr) {  
 return;  
 }  
 Print(node->left\_);  
 std::cout << node->key\_ << " ";  
 Print(node->right\_);  
 }  
  
  
 Node\* Search(Node\* node,T key) {  
 if (node->key\_ == key) {  
 return node;  
 } else if (key < node->key\_) {  
 return Search(node->left\_, key);  
 } else {  
 return Search(node->right\_, key);  
 }  
 }  
  
  
 Node\* root\_;  
};

*TreeArray.h*

#include <iostream>  
#include "Tree.h"  
template<typename T>  
class TreeArray:Tree<T> {  
 public:  
 TreeArray(size\_t size){  
 tree\_=new T(size);  
 size\_=size;  
 for (int kI = 0; kI < size; ++kI) {  
 tree\_[kI]= 0;  
 }  
 }  
 ~TreeArray()override{  
 delete[] tree\_;  
 }  
  
 void Insert(T key) override{  
 static size\_t count;  
 if(tree\_== nullptr){  
 T tree[size\_+1];  
 for(size\_t i=0;i<count;i++){  
 tree[i]=tree\_[i];  
 }  
 tree[count]=key;  
 for(size\_t i=count+1;i<size\_;i++){  
 tree[i]=tree\_[i];  
 }  
 }  
 else if(\*tree\_==0){  
 tree\_[count]=key;  
 }  
 else if(key<\*tree\_){  
 count++;  
 tree\_++;  
 Insert(key);  
 }  
 else if(key>\*tree\_) {  
 count++;  
 tree\_++;  
 tree\_++;  
 Insert(key);  
 }  
 else *//key==root->key\_)* return;  
 };  
  
 bool Search(T key)override{  
 for(size\_t i=0;i<size\_;i++){  
 if(tree\_[i]==key){  
 return true;  
 }  
 }  
 return false;  
 };  
  
 void Print()override{  
 int i=0;  
 int branches=0;  
 while(i<size\_){  
 i=2\*i+1;  
 branches++;  
 }  
 int branches\_=branches;  
 while(branches!=0){  
 std::cout<<tree\_[2\*branches+2];  
 branches--;  
 }  
 std::cout<<tree\_[0];  
 while(branches\_!=0){  
 std::cout<<tree\_[2\*branches+1];  
 branches--;  
 }  
 };  
  
 private:  
 size\_t size\_;  
 T\* tree\_;  
};

*Main.cpp*

#include "TreeDList.h"  
#include "TreeArray.h"  
  
int main() {  
 TreeArray<int> tree(6);  
 std::cout<<"Enter the amount of elements: ";  
 int size,element;  
 std::cin>>size;  
 for(int i=0;i<size;i++){  
 std::cin>>element;  
 tree.Insert(element);  
 }  
 tree.Print();  
 std::cout<<"\nEnter the key of searched node:";  
 int key;  
 std::cin>>key;  
 bool search=tree.Search(key);  
 return 0;  
}